In the claims:

For the Examiner's convenience, all pending claims are presented below with changes shown in accordance with the mandatory amendment format.

1-9. Cancelled

10. (Currently Amended) A method of eliminating partial redundancy in a routine, the method comprising:

~~(A)~~ speculatively computing down-safety by ignoring ~~rarely taken~~ branches in a control-flow graph that are known to be infrequent; ~~and~~

~~(B)~~ computing up-safety using the results of the down-safety calculation to determine where operations are speculatively available;

using the down-safety and up-safety to determine instructions to be inserted into the routine that set components in a stack, the stack to handle cleanup instructions for the routine; and

removing edges from one or more exceptional paths in the routine to eliminate the partial redundancy in the routine, the edges to be removed having purposes that are performed by the instructions.

11. (Previously Presented) The method of claim 10, wherein the computation of down safety further comprises a lattice that distinguishes strict down-safety from speculative down-safety.

12. (Currently Amended) The method of claim 10, further comprising determining whether [[a]] the routine includes the cleanup instructions prior to computing the down-safety.

13. (Currently Amended) The method of claim 12, further comprising:

removing the edges from [[an]] one or more exceptional paths if the routine does not include cleanup instructions; and

inserting a prologue at the beginning of the routine.

14. (Previously Presented) The method of claim 12, further comprising building a cleanup tree and cleanup states for the routine.

15. (Currently Amended) The method of claim 14, wherein the cleanup tree and cleanup states represents ~~an exception handling (EH)~~the stack at points where an exception may be thrown.

16. (Currently Amended) The method of claim 15, wherein the cleanup tree and cleanup states assist in the ~~determine operations~~ determining instructions to be inserted into the routine that set components in the stack.

17. (Currently Amended) The method of claim 10, wherein computing down-safety comprises:

computing a down-safety transfer function for each of ~~a plurality of~~ the edges of [[an]] the one or more exceptional path; and

solving ~~executing~~ flow equations for the down-safety transfer functions to produce a downsafe map from vertices in the routine to corresponding Boolean values.

18. (Currently Amended) The method of claim 17, wherein computing up-safety comprises:

computing an up-safety transfer function for each of the ~~a plurality of~~ edges of [[an]] the one or more exceptional paths; and

solving ~~executing~~ flow equations for the up-safety transfer functions to produce an upsafe map from vertices in the routine to corresponding Boolean values.

19. (Currently Amended) The method of claim 18, further comprising utilizing the downsafe map and the upsafe map to determine the ~~inserting~~ instructions to be inserted into the routine that set components of ~~an exception handling (EH)~~ the stack.

20. (Canceled)

21. (Currently Amended) A method comprising:

~~(A)~~ representing [[the]] a program with a control-flow graph in which actions to take in event of an exceptional situation are represented by ~~explicit~~ exceptional paths in the graph;

~~(B)~~ analyzing [[said]] the program to determine at which points [[the]] a stack must be in a valid state, the stack to represent cleanup instructions for the program;

~~(C)~~ building a forest of [[tress]] trees that represent [[the]] a stack state of the stack at said points where:

~~(i)~~ each node of [[the]] a tree of the forest of trees represents a possible item on the stack, and

~~(ii)~~ [[a]] the stack state is represented as a path from [[a]] the tree node to the root of the tree;

~~(D)~~ placing ~~computing where to place~~ operations in the program that set the state of items on the stack based on both of a down-safety calculation and an up-safety calculation for the program; ~~by performing the steps of:~~

~~(i) constructing flow equations for down-safety of operations that set the state of items on the stack, where the equations ignore rarely taken branches,~~

~~(ii) solving said flow equations for down-safety of operations that set the state of items on the stack,~~

~~(iii) constructing flow equations for up-safety of operations that set the state of items on the stack, where the equations use the solution for down-safety to determine which setting operations are speculatively available, and~~

~~(iv) solving said flow equations for up-safety of operations that set the state of items on the stack,~~

~~(E) using the results of said flow equations to place instructions that maintain the stack;~~

~~(F)~~ removing edges from the control-flow graph which represent ~~said~~ the actions for the exceptional events; and

~~(G)~~ inserting a prologue at an entry to the control-flow graph that saves [[the]] an existing pointer to the top of the [[EH]] stack.

22. (New) The method of claim 21, wherein the down-safety calculation is determined by performing:

constructing flow equations for a down-safety of operations that set the state of items on the stack, where the equations ignore branches that are known to be infrequent; and

solving the flow equations for the down-safety of the operations that set the state of items on the stack.

23. (New) The method of claim 22, wherein the up-safety is determined by performing:

constructing flow equations for an up-safety of operations that set the state of items on the stack, where the equations use the solution for the down-safety calculation to determine which setting operations are speculatively available; and

solving said flow equations for the up-safety of operations that set the state of items on the stack.

24. (New) The method of claim 22, wherein the calculation of the down safety further utilizes a lattice that distinguishes strict down-safety from speculative down-safety.

25. (New) The method of claim 21, further comprising building a cleanup tree and cleanup states for the program.

26. (New) The method of claim 25, wherein the cleanup tree and cleanup states represents the stack at points where an exception may be thrown.

27. (New) The method of claim 26, wherein the cleanup tree and cleanup states assist in the placing of the instructions to be inserted into the program that set state of items on the stack.